Network Support for Privacy-Preserving Forensic Attribution

Mikhail Afanasyev, Tadayoshi Kohno†, Justin Ma, Nick Murphy†,
Stefan Savage, Alex C. Snoeren, and Geoffrey M. Voelker
University of California, San Diego and †University of Washington
{mafanasyev,jtma,savage,snoeren,voelker}@cs.ucsd.edu, {kohno,nickm}@cs.washington.edu

Abstract

Privacy-preserving forensic attribution is a new architectural primitive we propose that allows individual network packets to be attributed, post-hoc, to the physical machines from which they were sent. Importantly, while our architecture allows any network element to verify that a packet has a valid forensic signature, only a trusted authority is able to reveal the sender's identity. In this way, the privacy of individual senders is protected from serendipitous use, while criminal actors cannot presume anonymity. We have developed a prototype implementation, called Clue*, that demonstrates the fundamental feasibility of this approach while also illustrating the design challenges and opportunities in integrating this functionality with the network layer. We hope this work stimulates further technical investigations in this area, as well as broader political and sociological discussions on the criteria for network-based privacy-preserving forensic attribution and its ability to address the current tensions between the demand for strong privacy and the push towards greater, privacy-invasive forensic techniques.*

1. Introduction

Research in network security has traditionally focused on defenses—mechanisms that impede the activities of an adversary. However, paraphrasing Butler Lampson, practical security requires a balance between defenses and deterrence [36]. While defenses may block an adversary's current attacks, only an effective deterrent can prevent the adversary from choosing to attack in the first place. Creating such a deterrent, however, is usually predicated on an effective means of attribution—tying an individual to an action. In the physical world this is achieved by collecting concrete forensic evidence—DNA, fingerprints, writing samples, etc.—but there are few equivalents in the digital domain.

As Peter Steiner's famous New Yorker cartoon states, "On the Internet, nobody knows you're a dog." Indeed, functional anonymity is *implicit* in the Internet's architecture since the lowest level identifiers, network addresses, are inherently virtual and insecure. An IP address only specifies a topological location, not a physical machine, and is easily forged on demand. Thus, it can be extremely challenging to attribute an on-line action to a particular physical origin (let alone to a particular individual).

Unfortunately, without a meaningful risk of being caught, attackers are free to act repeatedly and with impunity. It is this circumstance that underlies the asymmetric nature of the modern computer security arms race. Attackers are free to improve their methods until they can break our defenses—leaving defenders forever in the role of catch-up. Conversely, reducing this expectation of anonymity *even slightly* can potentially disincent a wide range of criminal activity and improve the effective lifetime of defense mechanisms.

Compelling though this line of thinking may be, there is a natural tension between the need for attribution and users' desire for privacy. While the public generally appreciates that criminal acts should be subject to scrutiny, civil libertarians are considerably less sanguine about exposing identifying information as a matter of course. Indeed, a recently leaked document, allegedly of ITU provenance, lends credence to libertarians' fears by motivating the need for network-level "IP Traceback" capabilities via a desire to uncover the identity of anonymous political opponents [30]. This is but one such tension, and it seems evident that the time is particularly ripe to explore technical solutions that can balance the enforcement interests of the state and the privacy interests of individuals.

This paper focuses on achieving this balance by introducing a new network-layer capability—*privacy-preserving forensic attribution*. In particular, we propose a packet-level cryptographic signature mechanism that allows properly authorized parties to examine any packet—even those logged months prior—and unambiguously identify the physical machine that sent it. However, absent this express authorization, packet signatures do not expose any identifying information. Finally, we enforce the correct use of this mechanism by allowing any network element to verify the validity of the signatures they receive.

To make progress, this paper purposely walks a line between "blue-sky" architectural research and "incremental" pragmatic research. Our goal is principally to demonstrate "viability" and thus encourage wider consideration of this problem. To this end, while we have largely ignored the challenges of deploying a modified Internet protocol or developing appropriate high-speed cryptographic hardware, we *have* built a prototype system, called *Clue*,

to explore the practical engineering challenges of building a privacy-preserving forensic attribution capability. Indeed, this experience has been critically illuminating. In addition to exposing the architectural requirements of our approach, it has also forced us to confront the real challenges of the underlying cryptographic overheads. Surprisingly however, we have found that much of this overhead can be hidden or amortized through careful protocol design alone. Thus, even our untuned user-level software prototype adds less than 30 ms of latency to interactive traffic and achieves a bulk TCP throughput over 17 Mbps. Moreover, this throughput—significantly greater than a typical broadband access connection—is *receiver-limited* and aggregate server throughput can be considerably higher.

2. Motivating Scenarios

Forensic attribution would create a fundamentally new network layer capability and here we describe several use cases motivating its potential value.

Investigating and prosecuting criminals. Law enforcement officers routinely face the challenge of mapping between traffic received at some point in the network and the physical device of origin. Establishing this mapping would allow investigators to determine if the same device was used in multiple crimes, if a particular activity was perpetrated by a known device, and potentially to even track the location of a targeted device via IP geolocation.

For example, the U.S. Secret Service is actively seeking methods for determining whether specific Internet packets originated from devices belonging to known terrorists. In these cases the terrorists are mobile and anonymously connect to the Internet from wireless hotspots. A more concrete example is motivated by the following e-mail one of us received from a Corporal in the Rhode Island State Police Department:

> We are currently attempting to locate an international fugitive who is wanted for [a violent crime]. We have identified a Pocket PC device attached to the Internet which the [fugitive] is apparently using. . . .

While we are not in a position to discuss all the details of this case, one can consider the following model: the police have a single packet trace known to be from the fugitive's device (perhaps a threatening e-mail) and now seek to identify if other threatening messages are from the same device—thus identifying the fugitives current IP address and hence area of operation.

Finally, it is increasingly common to recover computer equipment when suspects are taken into custody. However, tying that computer to other on-line actions—especially beyond a reasonable doubt—can be quite challenging absent a strong forensic identifier. For example, suppose an on-line criminal gang communicates with each other via a chat room monitored by law enforcement. When one of their members is arrested, his laptop is seized as well. Prosecutors would like to tie the laptop directly to the suspect's messages on the chat room—thereby supporting a conspiracy charge—but the suspect claims they are mistaken ("No, no... 'mafiaboy' is someone else, I never typed those messages"). A strong forensic identifier would allow a recovered laptop to be directly and unambiguously be bound to particular messages logged by law enforcement.

Disclaiming traffic. A large fraction of residential wireless access points are left "open" for any nearby user to route packets through. Indeed, many users do so knowingly, in support of community sharing. As well, there are a range of new commercial enterprises, such as FON, whose business model depends on users' willingness to provide public wireless Internet access via their own networks. Unfortunately, the anonymity provided by such open access points also attracts those interested in committing on-line criminal acts. In effect, open access points allow criminals to launder their identities and focus any blame on the innocent access point owner instead. Indeed, aside from nuisance risk of mistaken identity, the legal liability associated with providing anonymous transit service is far from clear.

Even incremental deployment of privacy-preserving forensic attribution could help in these situations. Namely, a homeowner wishing to offer open service could configure his or her wireless access point to only route traffic containing verifiable forensic signatures. Thus, if an attacker uses the homeowner's wireless network to perform criminal activities, their packets will necessarily contain sufficient evidence to both exculpate the access point owner and identify the criminal themselves.

Evil twin access points. A growing threat for mobile wireless users is "evil twin" access points—criminal-controlled open access points at public locations [34]. When a user connects to an "evil twin" access point, that access point will typically prompt the user for a name and credit card number.

However, such behavior could be deterred if the perpetrator were at some risk of being identified. In particular, users could refuse to join wireless access points that did not attach verifiable forensic marks to their packets—thus providing an audit trail should the user later suspect an evil twin attack. With help from law enforcement, the physical machine perpetrating the attack could be identified and, further still, their attributability credentials could be revoked—preventing their participation in future evil twin attacks.

3. Background

The value of forensic attribution—using technical means to establish the presence of a person or object at a crime scene after the fact—has a long history in law enforcement

dating to the late 19th century.[1] Lacking an eyewitness to a crime, forensic methods often become a critical tool in any investigation.

3.1. The challenge of IP-based attribution

Internet crime poses a unique challenge for forensic attribution. Unlike physical evidence, such as fingerprints and DNA, digital objects are, *prima facie*, non-unique. There are a range of reasons for this. First and foremost, the Internet architecture places no technical restrictions on how a host generates packets, and thus every bit a packet contains, including the "source" IP address, can be trivially manipulated and is thus potentially suspect. Indeed, source address spoofing has long posed security problems such as anonymous port-scanning [22], anonymous denial-of-service attacks [41] and TCP connection hijacking [11]. To mitigate this problem, many routers now include a source address validation mechanism (e.g., reverse path forwarding) to block the transmission of some kinds of spoofed packets [23]. However, these mechanisms are inconsistently deployed (Beverly and Bauer recently performed measurements suggesting that at least 25% of Internet address blocks are still spoofable [12]) and, at best, limit the attacker to spoofing addresses within their own contiguous address prefix (which may contain many thousands of addresses).[2]

A more fundamental problem is that IP addresses are not unique identifiers *even* when used as intended. An IP address represents a topological location in the network for the purpose of routing; it does not specify a physical endpoint. In fact, it is common for protocols such as DHCP, NAT and Mobile IP to dynamically change the mapping between IP address and physical machine as part of their normal use. While some such mappings can be logged, it is common that even this data is retained only for limited periods (sometimes less than a week [44]).

Wireless access poses an even more insidious problem. Public Wi-Fi hotspots allow attackers to purchase anonymity on a cash basis, or simply "hijack" the session of an existing user [56]. Private access points lower this bar further as many users do not configure any method of access control.[3] Indeed, recent reports have identified criminal use of open Wi-Fi access points specifically for the purpose of enhanced anonymity [48].

Finally, in their recent study of SPAM sending, Ramachandran and Feamster show that as much as ten percent of SPAM is sent from transient IP address space that is temporarily advertised and then withdrawn [45]. In such an instance, an IP address may not even be routable after an attack is completed, let alone serve as an identifier.

We are hardly the first to make these points. Indeed, they have been raised repeatedly by forensic professionals [29], security researchers [38], and Internet industry leaders [46] alike. Aucsmith sums the situation up well: "The Internet provides criminals two of the most coveted qualities: anonymity and mobility" [6].

3.2. Related work

There has been a wide range of systems designed to detect and/or block IP source address spoofing [23], [26], [31], [37], [39] although even in their ideal embodiment none of these are foolproof. There has also been a long line of literature focused on tracing spoofed packets back to their source [47], [52], [53], [57]. These approaches, however, are motivated by the operational needs posed by denial-of-service attacks and therefore focus on delivering topological path information—an even more abstract property than an IP address.

Finally, while we are unaware of other attempts to provide network-level forensic attribution to physical hosts, there are a number of related research projects that make similar use of cryptographic mechanisms. For instance, the Clipper chip (and its data-oriented sibling Capstone) was designed to provide generic end-to-end confidentiality and integrity while still allowing key recovery for law enforcement via government escrow [43]. The effort failed for a number of reasons including the complications of a U.S.-centric escrow policy for an international Internet and the negative public image of a "big brother"-like government entity with arbitrary powers. We discuss some of these same issues in Section 8 as they relate to our system. Closer to our own work, are the source authentication systems of Liu et al. ("packet passports" [39]) and Andersen et al. ("Accountable Internet Protocol" [1], [2]). Like us, both make use of cryptographic identifiers. However, these systems are focused on ensuring the consistency and topological validity of the IP source address itself to prevent address spoofing and do not address either user privacy concerns nor the need for long-term physical linkage required for forensic attribution. Finally, Wendlandt et al. describe the use of in-packet public key signatures for the purpose of authorizing expedited routes (again for protection against denial-of-service) [55]. While the problem is quite different, we have mutual interest in the development of high-speed hardware implementations of public-key cryptography suitable for router implementations.

4. Design goals

The design space for providing attribution is large and reflects the many different uses that we place on identity and

1. Calcutta first systematicly used human fingerprints for criminal records in 1897, followed by Scotland Yard in Britain in 1901.

2. Luckily, source address spoofing makes it difficult for attackers to participate in two-way communications and thus is self-limiting in the threat it presents.

3. In a 2006 survey performed by RSA Security, between 21 and 28 percent of Wi-Fi access points found in the cities of London, New York and Paris used a default open configuration [49].

the broad range of capabilities and limitations imposed by technology. For example, packet traceback systems, such as BBN's SPIE, can attribute an individual packet to a particular set of routers, but only do so with probabilistic correctness and over operational time scales (minutes to hours) [52]. By contrast, we have chosen a point in the design space focused on the unique requirements of forensic use:

4.1. Basic requirements

- *Physical names*. We believe that attribution must provide a link to a physical object (e.g., the sending computer). A physical computer can have an associated owner and thus permits association via sales and maintenance records. Moreover, given continuous ownership, a physical computer may be reused in multiple attacks. Identifying this computer allows these attacks to be linked even if the physical computer is never recovered. Finally, a physical computer accretes *physical* forensic evidence as a side effect of use. Indeed, much of this paper was written on a laptop that contains extensive fingerprint evidence on the screen and, upon examination, a range of hair and skin samples underneath the keyboard. Were this laptop to be found, it could be unambiguously linked to this author via DNA or fingerprint comparisons.
- *Per-packet granularity*. We believe the best deterrence is provided when attribution is universal — applied equally to every packet. Moreover, by providing this capability at the network layer, attribution is transparently provided to all higher layer protocols and applications. Put another way, there is an inherent benefit in not tying forensic attribution to any particular higher-level network construct. We argue that forensic attribution is most effectively used when provided as a fundamental building block upon which arbitrary higher-level protocols, services, and applications can be built.
- *Unimpeachability*. While we would be satisfied if a new attribution capability simply offered *investigative* value for those pursuing criminals, it is our hope that any attribution mechanism be accepted as sufficiently accurate and trustworthy to provide *evidentiary* value in the courtroom as well. Thus, we argue for the value of strong cryptographic mechanisms that cannot be easily repudiated.
- *Indefinite lifetime*. In the Internet, as in the real world, many crimes are not detected until long after they are committed. Placing unnecessary restrictions on the time window for forensic discovery will undoubtedly be exploited by criminals to their advantage. For example, even today many on-line criminals are savvy about the practical investigatory delays imposed by different bi-lateral Mutual Legal Assistance Treaties and locate their data accordingly. Thus, we argue that it should be possible to examine a packet and unambiguously attribute its origin long after the packet is received, a time difference that may be months or even years.

These requirements bring us to an architecture in which each packet is self-identifying — that is, tagged with a unique non-forgeable signature identifying the physical machine that sent it. While such attribution does not definitively identify the *individual* originating a packet, it is the critical building block for subsequent forensic analysis, investigation, and correlation as it provides a beachhead onto the physical scene of the crime. We presuppose that sites with sufficient risk and/or value at stake will check such signatures, associate them with higher-level transactions and log them for enough time to cover their risk. Building such a capability is straightforward using conventional digital signatures and some form of public key infrastructure, albeit with some performance cost.

4.2. Privacy requirements

However, this basic approach would allow anyone receiving such a packet to attribute its physical origin. While this violation of privacy may seem minor to some, there is a history of vigorous opposition to exactly such a capability. For example, in January of 1999, Intel Corporation announced that new generations of its popular Pentium microprocessors would include a new feature: the Processor Serial Number (PSN). The PSN was a per-processor unique identifier that was intended as a building block for future security applications. However, even though this feature was completely passive, civil libertarians and consumer rights groups quickly identified potential risks to privacy stemming from an available globally unique identifier. Among the specific concerns raised in complaints filed with the Federal Trade Commission were the general expectation that Internet users have to anonymity, the chilling effect of privacy loss on a consumer's use of the Internet, the potential for a global identifier being used to cross reference personal information, and the potential for identity theft by forging the PSN of others [42]. In April 2000, Intel abandoned plans to include the PSN in future versions of its microprocessors. Thus, we posit another critical requirement:

- *Privacy.* To balance the needs for forensic attribution with the public's interest in privacy, packet signatures must be non-identifying, in a strong sense, to an unprivileged observer. Moreover, the signatures must not serve as an identifier (even an opaque one). As such, distinct packets sent from the same source must carry different signatures. Roughly speaking, users should have at least the same expectation of

anonymity that they have in today's Internet excepting authorized investigations.

On its face, a potential solution to this problem is to digitally sign each packet using a per-source key that is in turn escrowed with a trusted third party. Indeed, it was just such an approach that was proposed by the ill-fated Clipper chip. However, if a single third party is not widely trusted (which seems likely given past experience), then the scheme may accommodate multiple third-parties responsible for different sets of machines, and/or a secret sharing approach in which multiple third parties must collaborate to generate the keying material to validate the origin of a signature (e.g., in the U.S., both the Department of Justice and the American Civil Liberties Union might be required to agree that an investigation is warranted). There is, however, a critical vulnerability in this approach. Since, by design, a normal observer cannot extract any information from a packet signature, nothing prevents an adversary from incorrectly signing their packets (i.e., with random signatures). In such a situation, any attempts at post-hoc authentication will be useless. Thus, to be practical, our attribution architecture is motivated by one final requirement:

- *Attributability.* To *enforce* the attribution property, any observer on the network must be empowered to *verify* a packet signature. That is, an observer should be able to prove that a packet *could* be attributed if necessary, but the process of performing that proof should not reveal any information about the physical originator itself. This requirement has a natural fate-sharing property since the choice to verify a packet is made by the recipient with the most future interest in an attribution capability.

4.3. Non-goals

Equally important as our design goals, are our non-goals — what we do not hope to accomplish in this paper. For one, our work is not designed to address the issue of IP address spoofing. While there is operational value to preventing spoofing (i.e., to allow easier filtering of DDoS attacks), in our view the virtual nature of IP addresses make them inherently ill-suited for forensic purposes. Second, we wish to be clear that our work in this paper is strictly limited to attributing the physical machine that sent a particular packet and not necessarily the complete causal chain of events leading to that packet being generated. This distinction is common to most kinds of forensic investigations (e.g., unraveling offshore shell accounts in forensic accounting or insider communication in securities fraud investigations) but can manifest particularly easily in the Internet context. For example, an attack might be laundered through one or more intermediate nodes, either as part of a legitimate anonymizing overlay network (e.g., Tor) or via proxies installed on compromised hosts. Previous work has explored how such "stepping-stone" relations may be inferred in the network [58] and we believe that a similar approach — attributing each causal link hop-by-hop — could be employed with our architecture as well (and with the benefit of unambiguous evidence of a packet origination). However, we believe that unambiguously establishing such causality is not possible at the network layer alone and will ultimately require both host support and, inevitably, manual investigation as well. While we have a vision for how such host services should be structured, we consider them beyond the scope of this paper.

5. Architecture

While the previous requirements appear quite challenging to satisfy, surprisingly there is a well-known cryptographic tool — the *group signature* — that neatly unties this particular Gordian knot. In this section, we briefly review the properties of group signatures, describe their basic application to forensic packet attribution, and review the design issues that result from this architecture.

5.1. Group signatures

Group signatures were first introduced by Chaum and van Heyst [19] with security properties fully formalized in papers by Bellare, Micciancio, and Warinschi [8] and Bellare, Shi, and Zhang [10]. We informally describe their operation below using the model of [8], as well as a definitional extension due to Boneh, Boyen, and Shacham [13].

Informally, a *group signature* provides the property that if a member of a group signs a message, anyone can verify that the signature was indeed created by *some* group member, but they cannot determine *whom* without the cooperation of a privileged third party known as the *group manager*.

More formally, a group signature scheme assumes a group manager and a group of n unprivileged members, denoted $1, 2, \ldots, n$ (in this text we assume n is known, but [10] extends the model to include groups of dynamic size). The group manager has a secret key msk, each group member $i \in \{1, \ldots, n\}$ has its own secret signing key $\mathsf{sk}[i]$, and there is a single public signature-verification key pk.

The group manager uses a KeyGen operation to create $\mathsf{pk}, \mathsf{msk}, \mathsf{sk}$ and distributes these appropriately. Subsequently, if a group member i uses its secret key $\mathsf{sk}[i]$ to Sign a message m and get back a signature σ, anyone with access to the signature-verification key pk can Verify that (m, σ) is a valid message-signature pair under the secret key of *some* group member. The group manager can use msk to recover the identity i of the signer using the Open operation.

There are two principle security properties for group signature schemes: full-anonymity and full-traceability;

these properties imply other properties, including unforgeability, exculpability and framing-resistance [8]. A group signature scheme is *CCA-fully-anonymous* [8] if a set of colluding members cannot learn information about the signers' identity i, even when the adversaries are allowed to Open the signatures for all the messages besides the target message-signature pair. Boneh et al. define a slightly weaker definition of anonymity, *CPA-full-anonymity* [13], in which the adversary is unable to Open the signatures on other message-signature pairs. Surprisingly the set of colluding members can actually include i, such as if i were coerced into cooperating with the adversary. Similarly, a group signature scheme is *fully-traceable* [8] if a set of colluding members cannot create a valid message-signature pair (m, σ) that the group manager cannot trace back to one of the colluding parties — i.e., either $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ fails, meaning that the signature is invalid, or $\mathsf{Open}(\mathsf{msk}, m, \sigma)$ returns the identity of one of the colluding members.

5.2. Basic packet attribution

We apply group signatures to our problem as follows. Each machine is a member of some group and is provided with a secret signing key. Exactly how groups are constructed is very much a policy issue, but one pragmatic approach is that each computer manufacturer defines a group across the set of machines they sell. This is a particularly appealing approach because it side-steps the key distribution problem, as manufacturers are now commonly including Trusted Platform Modules (TPM) that encode unique cryptographic information in each of their machines (e.g., the IBM Thinkpad being used to type this paper has such a capability). Moreover, a tamper-resistant implementation is valuable to prevent theft of a machine's signing key. This would also imply that the manufacturer would either act as group manager in any investigations (i.e., will execute Open under subpoena) or would escrow their msk (or shares thereof) to other third parties. However, this particular decision is not critical to our technical discussion and we defer further discussion of the surrounding policy issues until Section 8.

Given a secret signing key, each machine uses it to sign the packets that they send. This signature covers all *non-variant* protocol fields and payload data. The name of the group and the per-packet signature are included in a special packet header field that is part of the network layer. Any recipient can then examine this header, and use Verify to validate that a packet was correctly signed by a member of the associated group (and hence *could* be authenticated by the group manager). The verify step does not require the use of any protected key material and, by virtue of fate sharing, need not be part of the trusted computing base.

5.3. Design issues

An implementation of group-signature-based packet attribution must address several other challenges before deployment becomes practical.

Replay. The basic approach we have outlined does not prevent the adversary from replaying messages sent (and signed) by other legitimate parties, or shuffling the order in which a node receives the messages from other legitimate parties. In some cases such replayed packets will be immediately discarded by the receiving protocol stack or application (e.g., because of misaligned sequence numbers or routing information). On the other hand, an adversary might be able to mount an untraceable DoS attack or maliciously change application behaviors by replaying or shuffling others' packets over time. We therefore desire some mechanism to bind these packets to a particular point in time.

One natural solution requires the sender to include a monotonically increasing counter in each packet, and the receiver to discard any packets with duplicate sources and counters. However, the naive implementation of such an approach may require the receiver to maintain the list of source-counter pairs (i.e., through reboots). Our current implementation assumes loosely synchronized clocks; the signer includes the current time in each outgoing packet, and the receiver validate freshness directly. To handle jitter and various network delays, as well as possible inconsistencies between different devices' perceptions of time, one might employ a hybrid approach, including both a counter and the time in each packet.

Revocation. To ensure that verifiable packets are attributable back to a single physical machine, we assume that the group signature secret keys will be stored in tamper-resistant hardware and will not be copyable to other devices. However, we anticipate that some secret signing keys will inevitably be compromised. There are two general frameworks for revoking these secret signing keys — the first by Camenisch and Lysyanskaya [17] and the other by Brickell and others [5], [14], [16], [32]. Since most parties in our system are both signers and verifiers, we adopt the Camenisch-Lysyanskaya approach in which secret keys are revoked by globally updating the group public key and locally updating each un-revoked party's secret signing key. In this scheme, the verifiers do not need to maintain a list of individual revocations, but public key updates must be applied universally to ensure that all subsequent signatures can be verified. If a device misses an update, then it must obtain those updates through a back channel before its packets can be verified under the updated group public key.

Middlebox modification. Middleboxes, like network address translators (NATs), create a conundrum. In our architecture, senders sign all non-volatile [52] contents of

outgoing packets, *including* the source address. Thus, any packets that traverse a NAT will no longer verify, as their contents have been changed. While some may consider this a shortcoming, we argue that it is a requirement of true attribution — the signer can only attest to contents she transmitted. The only other option in our framework is to deem the source address volatile, and exclude it from the packet signature. To do so would imply that the source address has no significance beyond a routing locater; unfortunately, that is not the case in the current Internet: end hosts use source addresses to demultiplex incoming transport connections as well as associate flows with the appropriate IPsec security associations.

This tension has been observed many times in the past, and two architecturally pure alternatives exist: future Internet architectures can either remove end hosts' dependence on IP source addresses [28], [24] or make the presence of middleboxes explicit [54]. For the time being, deployments requiring NAT-like functionality must make a tradeoff between deployability and completeness: they are forced to choose between removing source addresses from the signature — thereby limiting the scope of the attribution — or encapsulating the original, signed packets in an IP-in-IP tunnel [51], thereby exposing the middlebox to the receiver.

On the choice of group signature schemes. Numerous group signatures schemes have been proposed following the seminal work of Chaum and van Heyst [19], each with different security or performance properties. Since our ultimate goal is to evaluate the viability of our overall approach from a holistic perspective, assess the network implications and challenges, and stimulate further research, our particular choice of group signature schemes is somewhat less material, and indeed we expect future innovations in cryptography to produce even more attractive schemes. Foreshadowing to Section 6, our prototype leverages the Boneh, Boyen, and Shacham group signature scheme [13], which is provably CPA-fully anonymous and fully-traceable under well-defined assumptions. Our choice was driven in part because of its short group signature size and its support under the Stanford Pairing-Based Cryptography (PBC) Library [40]. While CPA-full anonymity is a weaker property than CCA-full anonymity, it is reasonable in many cases [13], and especially for ours since we expect for the opening of signatures to be a rare process tightly controlled by a coalition of multiple parties. Other group signature goals, such as strong exculpability [3], [4], [10], [33] (where even the issuer of secret keys cannot forge signatures) do not seem necessary for our model since the issuer of a private key might also be the manufacturer of the hardware; nevertheless, it would be possible to make our system strongly exculpable using the enhancements in [13].

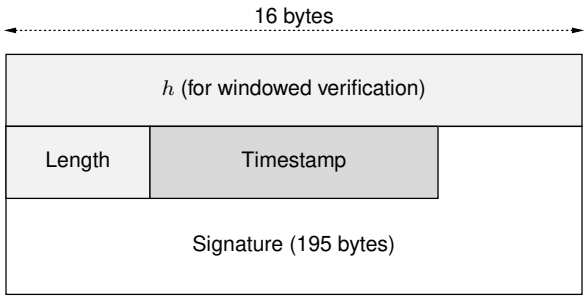

Figure 1. *Clue* packet trailer format. The shaded fields are explained in Section 6.2.

6. Clue

To explore the systems issues related to implementing a real-world group signature scheme, we have developed a prototype called *Clue*. *Clue* uses Stanford's Pairing-Based Cryptography (PBC) Library [40] for group signature operations, which in turn uses the GNU Multiple Precision arithmetic library (GMP). In order to explore group signatures in the context of real network packets, we have implemented *Clue* as a module in the Click Modular Router [35]. As PBC and GMP are designed as user-mode libraries, we run Click as a user-mode process rather than as a Linux or BSD kernel module. While this does incur some performance penalty on its own, the cryptographic operations dominate the user-mode transitions in practice, and the user-mode penalty does not interfere with the fundamental system design issues.

Figure 1 shows the packet trailer used by our current prototype. The *Clue* module is mostly a straightforward packet transformation element. When signing, the module:

1) Collects the non-volatile elements of an IP packet
2) Adds an eight-byte, local NTP-derived timestamp to implement replay detection
3) Feeds the resulting data as input to the group signature library to generate a signature, and
4) Appends the signature (and additional optimization information) to the original packet and adjusts the IP length field accordingly

Tasks such as recalculating checksums are left to other, standard Click elements in the pipeline that perform these functions. Similarly, when verifying, the module:

1) Validates a packet's freshness from its timestamp
2) Collects the non-volatile elements of an IP packet
3) Strips the *Clue* trailer from the end of the packet
4) Feeds the resulting data and signature to the group signature library, and
5) Pushes the original packet to one of two output ports depending on whether verification was successful

We implement Boneh et al.'s revocation scheme [13], driven by polling a well-known revocation service.

As might be expected, cryptographic operation overhead can be quite high and dominates most performance

measures. While we have little doubt that more efficient group signature schemes will emerge with faster implementations (we note that [55] describes a software-based public-key implementation with sub-100 microsecond verify times) and that hardware implementation provides further capacity, in this paper we have focused on the optimization opportunities that arise from the interaction between the dynamics of network protocols themselves and the underlying cryptographic primitives. We first describe the particular group signature construction we use and then a series of optimizations we have implemented.

6.1. BBS Short Group Signatures

Our prototype uses the Boneh, Boyen, and Shacham [13] *short group signature scheme*, which exhibits comparatively short signatures relative to group signature schemes based on the Strong-RSA assumption of Baric and Pfitzmann [7]. We also refine the BBS group signature scheme for use with our optimizations. The following summarizes the basic BBS scheme at a level sufficient to understand our optimizations; we defer further details of the BBS scheme to the Appendix and reference [13].

The BBS Sign algorithm, on input a group public key pk, the signer's secret key $\mathsf{sk}[i]$, and a message m, first obtains a source of randomness $\mathcal{V}$, derives values for the variables $T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5$ from $\mathcal{V}$ and pk, and then computes the value c as $c \leftarrow H(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ where $\leftarrow$ denotes assignment from right to left and H is a hash function; for the security proofs, BBS model H as a random oracle [9], [13]. The signing algorithm then outputs the signature

$$\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}) \,, \qquad (1)$$

where $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}$ are functions of $c, \mathcal{V}$, and $\mathsf{sk}[i]$. The BBS Verify algorithm, on input a group public key pk, a message m, and a signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$, derives $R'_1, R'_2, R'_3, R'_4, R'_5$ from pk and σ, computes c' as $c' \leftarrow H(m, T_1, T_2, T_3, R'_1, R'_2, R'_3, R'_4, R'_5)$ and accepts the signature as valid exactly when $c = c'$. None of our optimizations or extensions modify the BBS KeyGen or Open algorithms; we therefore do not survey their details here.

6.2. Optimizations

There are several approaches for optimizing a BBS-based packet attribution system. These optimizations exploit artifacts of the BBS scheme itself as well as properties of networks protocols and the clients.

Precomputation (for sender). We can take advantage of client workloads to improve the overhead of Sign in the sending critical path. The Sign operation has two components, computing the T_j and R_j values, which are independent of packet data, and using those values to sign a packet. The T_j and R_j computation step by far dominates the overhead of Sign. If we take the T_j and R_j computation out of the critical sending path by precomputing them, we can greatly improve the throughput of using Sign. Most client workloads consist of applications that have low average sending rates, such as email, Web browsing, remote login, etc., allowing signature precomputation to overlap I/O. Indeed, over long time scales the CPU load of clients and servers alike are dominated by idle time — an effect further magnified by multi-core processors. Thus, periods of idleness can be exploited to buffer signature precursors for subsequent periods of activity [27]. As we will show, this precomputation dramatically reduces the requirements of the sender and verification performance becomes the dominant bottleneck. Thus, we have not explored additional performance improvements for the sender; e.g., scenarios in which the sender only signs packets in batches, perhaps producing only one signature every k packets.

Windowed verification (for receiver). We take advantage of the streaming nature of network protocols like TCP to amortize verification over multiple packets of data to reduce the overhead of Verify in the receive critical path. Deriving the R'_j values from pk and σ creates a significant fixed overhead for Verify independent of the amount of signed data. When using Verify on the receiver, the attribution layer can accumulate a window of packets (e.g., a flight of TCP segments) and verify them all together to amortize the per-packet verification overhead. We stress that the signer signs every window of k packets, even overlapping windows, and that the verifier has the option of either verifying the packets individually, or verifying any window of its choice. However, this verification optimization slightly increases the length of a signature.

To accommodate the above scenario, we modify the BBS scheme as follows. Using our modified scheme, a verifier can choose to verify the signature on the j-th packet P_j in isolation (e.g., when there are no other packets waiting to be verified or when there is packet loss), or verify in batch the signature on a window of k packets $P_{j-k+1}, \ldots, P_j$. We accomplish the above goal by, on the signing side, first hashing the initial $k-1$ packets $P_{j-k+1}, \ldots, P_{j-1}$ to a value h, then signing $h \| P_j$ as before, and finally including h in the resulting signature tuple; here $\|$ denotes string concatenation, the hash function to compute h is $H' \neq H$, and P_j is implicitly prefixed with a fixed-width length field. To avoid trivial hash collisions in h, when hashing the packets $P_{j-k+1}, \ldots, P_{j-1}$ we also prepend each packet with a four-byte length field, and then concatenate the resulting length fields and packets together. Including h in the signature allows for the receiver to verify the signature over the j-th packet P_j in isolation (by verifying the signature

over $h \| P_j$). To verify the signature over the entire window $P_{j-k+1}, \ldots, P_j$, the receiver first recomputes h.

Security for the above construction follows from the proof of security for the basic BBS group signature scheme [13] assuming that H and H' are independent random oracles and that the hash values h are always the same length. In practice, since the inputs to H and H' are prefix-free [21], one can implement H and H' from the same hash function, like SHA-1 or SHA-512, by prepending the inputs for H with a 0 byte and the inputs for H' with a 1 byte.

In our implementation the window size k is a parameter provided to the IP layer. While it can be set statically, we have also modified our TCP implementation to adaptively set k to match the sender's congestion window. We believe this maximizes performance since it reflects the largest number of packets that can be amortized together without expecting a packet loss (losing the benefit of amortized verification).

Asynchronous verification (for receiver). Furthermore, we can overlap computation with network delay to reduce protocol serialization with verification. For example, rather than wait until Verify completes on a TCP packet before sending an ACK, TCP can first optimistically send an ACK back to the sender to overlap the ACK with the Verify computation. Implementing this feature is inherently a layer violation since we allow TCP ACK processing to proceed independent of IP layer verification, but we prevent unverified data packets from being passed to the application.

Incremental verification (for receiver). Given the computational costs associated with the Verify algorithm, under some circumstances, e.g., DoS attacks, we may wish to be able to quickly reject packets that might not be attributable. While we currently cannot completely erase the cost for verification, we can decrease the amount of time to reject a non-verifiable packet by a factor of approximately 2, at the expense of increased signature sizes. The approach we take is to make Verify incrementally verifiable — the average time to process and reject a non-attributable packet will decrease, though the time to process and accept a legitimate packet will remain essentially unchanged.

Our incrementally verifiable version of the BBS group signature scheme builds on our earlier observation that (1) the bulk of the computation in Verify is spent on computing $R'_1, \ldots, R'_5$ and (2) an implementation can derive $R'_1, \ldots, R'_5$ in parallel. Technically, we change the signature output in Equation 1 to

$$\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}, R_1, R_2, R_3, R_4, R_5) \ .$$

We then revise the Verify algorithm to, on input a signature σ, set $c'' \leftarrow H(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$, and immediately reject if $c'' \neq c$. The modified verification algorithm would then derive the variables $R'_1, R'_2, R'_3, R'_4, R'_5$ from pk and $T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}$ in a *random* order, immediately rejecting if $R'_j \neq R_j$. Finally, the modified algorithm would accept the signature as valid since the failure to reject above implies that $c = H(m, T_1, T_2, T_3, R'_1, R'_2, R'_3, R'_4, R'_5)$.

This version of Verify is incrementally verifiable since it is possible for Verify to reject before deriving all of $R'_1, \ldots, R'_5$. An adversary cannot force the verification algorithm to always take the maximum amount of time since the equalities $R_j = R'_j$ are checked in a random order unknown to the adversary. This construction also preserves the anonymity and traceability properties of the basic BBS scheme. Intuitively, since anyone could compute the $R'_1, \ldots, R'_5$ values from the signature itself, including the R'_j values in the signature does not break the anonymity property. Additionally, since the T_j values are untouched and the values R'_j and R_j are always checked for equality, this variant preserves the BBS traceability properties. In the common case, this modification does nothing except consume extra bandwidth (since Verify still needs to compute $R'_1, \ldots, R'_5$). But, in the event of an attack consisting of a set of non-verifiable packets, this modification allows us to more quickly discard a fraction of those packets.

Potential optimizations. There is a large class of related optimizations that relax security guarantees in exchange for performance. For example, the receiver could randomly verify a packet with probability $1/n$ for some n. However, we have explicitly chosen not to explore such optimizations at this time since our goal with this paper is to explore an extreme point in the design space — one where the attributability of each and every packet can be enforced. For the same reasons, we have not yet explored protocol-specific fate sharing optimizations such as only signing and verifying TCP SYN packets. Such optimizations could dramatically reduce overhead, albeit in exchange for some increased risk of non-attributability (e.g., via TCP connection hijacking).

Another possibility for improving performance is to investigate parallel hardware implementations of the cryptographic operations; such hardware is likely appropriate at least for router implementations and heavily-loaded servers. For example, this hardware could contain multiple cores, each implementing a complete version of the BBS signing and verification algorithms. Or one could exploit the fact that, internal to Sign and Verify, the computations of the T_js, the R_js, and R'_js are independent and parallelizable; see Equations 2–4, 5–9, and 16–20 in the Appendix. Rather than explore such forms of parallelism in this paper, we choose to take advantage of properties of

	1 packet	8 packets
sign	6.17	6.19
precomputed sign	0.022	0.058
precomputation	6.14	6.14
verify	15.7	15.7
incremental verify	15.6	16.3
corrupted incremental verify	4.85	4.83

Table 1. Overheads (in ms) of cryptographic operations for both 1 and 8-packet windows.

networking protocols and client behavior to improve the performance of software implementations.

7. Evaluation

We have verified that *Clue* provides the security properties described in Section 4. Here, we quantify the overhead of the *Clue* implementation of the basic security operations, and we measure the impact of this overhead on TCP performance. We emphasize, however, that benchmark figures are presented to demonstrate the feasibility of our approach—in particular, we have not optimized the cryptographic operations.

As a user-level software prototype, *Clue* provides acceptable performance when using the optimizations described in Section 6.2. *Clue* adds about 30 ms of end-to-end delay to sending a packet. For interactive applications like `ssh`, this extra delay is insignificant to users. *Clue* achieves a bulk TCP throughput of 17.5 Mbps—a throughput greater than that enjoyed by the average wide-area Internet user. A typical Internet user browsing the Web using *Clue* would experience roughly the same performance as without.

7.1. Experimental setup

For our experiments, we use three hosts in a sender-delay-receiver configuration. The delay host ran Linux 2.6 with a hardware configuration of dual 2.8-GHz Pentiums with 2 GB of RAM, and ran the NIST Net emulation package [18] to introduce link delay in our TCP experiments. The sender was a dual 3.4-GHz Pentium with 4 GB of RAM, and the receiver ran dual 3.0-GHz Pentiums with 16 GB of RAM. Both the sender and receiver ran the Click-based implementation of *Clue* (see Section 6) over Linux 2.6 and used the default values for send and receive buffer sizes.

For all experiments, we use the `d277699-175-167` parameter file pre-packaged with PBC, which yields a group signature scheme with strength roughly equivalent to a standard 1024-bit RSA signature [13]. The BBS scheme outputs signatures that are 195 bytes long using this parameter file.

7.2. Microbenchmarks

We start by measuring the overhead of the basic cryptographic operations Sign and Verify and their variants as described in Section 6. Table 1 shows the average time taken across 100 iterations of these operations on the receiver. The first column of results for "1-packet sign window" show overheads when executing on a single 1277-byte packet as input; we choose 1277 since the combination of such a packet, 195 bytes for the basic BBS signature, 8 bytes for the timestamp, and an extra 20 bytes for the windowed signing optimization will yield a 1500-byte packet. The second column, "8-packet sign window," shows results with eight packets as input; as described in Section 6.2, one of our optimizations uses windows of packets to amortize overhead across multiple packets. In either case, the per-packet overhead is sufficiently small (10–30 ms in total) to be unnoticeable in interactive traffic, but substantial enough to have a significant effect on bulk TCP performance.

The precomputation optimization for the sender separates signature computation from signing the packet. The "precomp sign" result measures the step that remains on the critical path—signing using a set of precomputed values—and shows that almost all of the overhead of Sign comes from generating message-independent cryptographic values (the "precomputation" step) and *not* from computing the message-dependent part of the signature nor signing the packet itself. In the bulk-transfer experiments, we show that removing signature computation from the critical path of sending packets results in a significant increase in throughput. Similarly, the row labeled "verify" represents the average time to verify a single signed packet of the same size listed above; in our implementation, verification is about 2.5x slower than signing.

The remaining two rows measure the performance of our incremental verification scheme designed to defend against a flood of invalid packets described in Section 6.2. The "incremental verify" row shows the time required to verify a valid packet signature using this scheme, which is essentially identical to the original verification operation, introducing negligible CPU overhead in the common case. In contrast, "corrupted incremental verify" measures the average time required to reject a corrupted signature. For the original scheme, this time is identical to the "verify" measurement. However, using incremental verification we obtain a 70% reduction in overhead.

The only significant difference between the eight-packet times and the single-packet times occurs when signing a packet using pre-computed values and arises as a result of hashing the extra data in the additional packets. Note, however, that this cost is still roughly two orders of magnitude less than any other operation, and, as such, we do not observe any additional impact on bulk throughput. As a result, amortizing the attribution operations over multiple packets is a key mechanism for reducing receive overhead. In our experiments below, we show that large windows combined with precomputed signatures can dramatically improve performance over basic Sign and Verify alone.

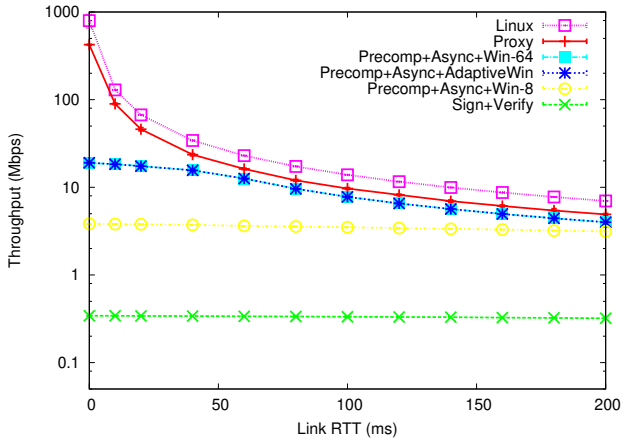

Figure 2. Throughput performance for TCP for combined optimizations. The y-axis is shown in log scale.

While not time-critical, for completeness we have also measured the performance of key revocation. It takes our *Clue* prototype approximately 21.2 ms to recompute a group member's private key on our test hardware. Recall that each un-revoked group member must recompute her private key each time the public group key is updated. The relative speed of this operation, however, indicates that the coordinated distribution of group key updates—as opposed to any cryptographic operations—is likely to be the limiting factor on key revocation rates.

7.3. TCP throughput

Bulk TCP throughput is an important performance metric for many Internet applications. Experimentally, our goal is to evaluate the impact of attribution on TCP throughput in our *Clue* prototype. We measure the TCP throughput performance of the our attribution implementation relative to various baseline configurations across a range of network round-trip times (RTTs). The implementation of attribution in *Clue* achieves a throughput within a factor of 1.2 of a Click forwarder at typical Internet RTTs. While privacy-preserving attribution has a non-negligible impact on bulk throughput on today's client systems, this cost is not prohibitive and will continue to decrease over time as CPU performance increases faster than typical Internet bandwidths.

We conducted `ttcp` benchmarks between the sender and receiver that required those hosts to forward traffic through a delay host. For every test configuration, we ran each individual transfer for at least 20 seconds. We require the sender to transfer all its data before it closes the connection, and we time the transfer from when the sender connects to the receiver to when the sender receives the FIN from the receiver. Figure 2 shows the results of our experiments for a number of different configurations. We vary the round-trip time (RTT) between sender and receiver on the x-axis, while the y-axis plots the throughput achieved using the `ttcp` application benchmark; note that the y-axis is a log scale. Each point is the average of five runs; error bars show the standard deviation.

As an upper bound, the "Linux" curve plots the forwarding rate of the default Linux networking stack on our hardware. To provide a more realistic baseline for our implementation, we also show the performance of an unmodified user-level Click installation ("Proxy"); Click simply forwards packets received on its input to its output without any processing. The difference between "Proxy" and "Linux" shows the overhead of interposing in the network stack at user-level, including the copying overhead when crossing the kernel boundary, etc. An optimized, kernel-level packet attribution implementation need not suffer this overhead. Although not shown, we also measured the performance of the provided Click IPsec module and found its performance to be indistinguishable from the "Proxy" configuration.

The "Sign+Verify" line corresponds to the baseline performance of *Clue* using individual Sign and Verify on each IP datagram. Given the times required for Sign and Verify as shown in Table 1, one would expect the 29 ms required for the Verify operation to limit long-term bulk throughput to a maximum of 0.35 Mbps. It is not surprising, then, that our implementation of the default "Sign+Verify" attribution process restricts bulk TCP throughput to approximately 0.33 Mbps independent of the RTT.

The poor performance of "Sign+Verify" motivates a need for the optimizations described in Section 6.2. While precomputation dramatically decreases the overhead the sender, it has only modest impact in isolation on TCP throughput, as performance is still receiver limited. Similarly, asynchronous verification allows the receiver to immediately issue ACKs, but the potential for improvement is bounded by the effective decrease in flow RTT. Indeed, precomputation and asynchronous verification are most effective when combined with windowed verification, which has the potential to move the performance bottleneck back to the sender.

The line titled "Precomp+Win-8" shows the performance of *Clue* when combining the three optimizations while using a fixed window size of eight packets. In theory, the larger the window size, the less overhead verification imposes. Indeed, progressively increasing the window size continues to increase throughput performance—to a point (most of the benefits have been achieved with a window of 64 packets as indicated by the line "Precomp+Async+Win-64," which exceeds 17.5 Mbps at 20 ms). Recall that windowed verification can only proceed in the absence of loss; if a packet is lost in a window, the remaining packets must be verified individually, negating any potential for improvement. Hence, our *Clue* implementation dynamically adjusts the window

size to match the sender's TCP congestion window. The "Precomp+Async+AdaptiveWin" line shows its performance approaches the baseline for all but the smallest RTTs. In fact at an RTT of 80 ms—a typical RTT for TCP connections on the Internet [50]—this combination achieves a throughput of 9.6 Mbps, within a factor of 1.2 of "Proxy" itself, and exceeds the capacity of most consumer broadband links.

8. Technology and politics

We have so far largely evaded the policy question of who should play the role of group manager and under what circumstances a packet "opening" should be authorized. This is not simply because it has no technical answer, but because it clearly has no single best answer. The Internet is an international entity and one without any overarching controlling legal authority. Whose laws should apply in authorizing attribution? What parties should be involved, and how? It is inevitable that different prevailing standards will hold sway and may ultimately require treaty instruments to resolve.[4] At the same time, there are choices in a technical design that can have profound effects on the manner in which attribution is used or abused. It is, as David Clark *et al.* describe, a classic example of a "tussle" — a point of contention between the interests of law enforcement, individuals, governments and industry [20]. However, rather than completely ignore this hard issue, we offer a strawman proposal and argue why it offers a useful point for debate.

First, as discussed in Section 5, since we assume that individual group signing keys are installed into per-computer TPMs it would be expedient to have the manufacturer play the role of group manager. Thus each packet would include not only the packet's signature but also a group identifier associated with the manufacturer.[5] This avoids the complexities and political complications of a PKI and is also a natural role for manufacturers since they are increasingly multi-national entities who understand local jurisprudence. Moreover, manufacturers already maintain records of sale and service that would inevitably be requested anyway after a packet is attributed to a particular machine. While it is certainly possible that some manufacturers may be compromised or may seek jurisdictional protection from legal subpoena by incorporating in uncooperative nations, this behavior is ultimately visible and thus potentially self-policing. If large numbers of Internet hosts or networks refuse packets from questionable or non-responsive manufacturers then their products are unlikely to succeed in the marketplace.

However, another concern is that corporations may voluntarily share data with government entities (or even third party corporations) due to extra-legal pressure or simple economic interest. In a sense this is unavoidable in general and cannot be solved strictly through technical means. However, within compatible systems of government, it is *possible* to build checks and balances that mitigate this threat and make such actions transparent — to watch the watchmen as it were. To wit, suppose that after generating a stockpile of group keys, the group manager private key is split into n pieces that are doled out to independent oversight organizations.[6]

Attribution can then be performed using standard cryptographic techniques for threshold decryption [25]. In particular, any t out of n organizations may combine their pieces to perform the Open operation on a particular signature, but without any of the organizations recovering the original private key. Thus, a packet cannot be attributed without the participation of t parties. This permits a range of policy choices. The oversight organizations can stymie extra-legal attribution by requiring a court order. Alternatively, they may publish any attribution requests so the state's use of this technology and their particular interests can be scrutinized. Again, this approach is itself imperfect within a coercive state or against a set of collaborating rogue oversight organizations, but we believe it is a useful starting point for discussion.

9. Conclusions

Much of the Internet's success can be attributed to its minimalist architecture. By asserting few restrictions, the same network substrate has allowed connectivity across heterogeneous communications technologies and has supported a tremendous variety of application uses. However, these architectural choices were made without significant consideration of the security implications. Indeed, in David Clark's classic paper "The Design Philosophy of the DARPA Internet Protocols", the word *security* does not appear. Today, the Internet's architectural freedoms have emerged as ripe vulnerabilities for adversaries trying to exploit the network to their ends. Chief among these is the lack of accountability for user actions. Without a plausible threat of accountability, the normal social processes that disincent criminal behavior cease to function.

In this paper we consider the possibility of modifying the Internet architecture to proactively enable network forensics while still preserving the privacy of network participants under normal circumstances. Our approach is to use a tool from cryptography—group signatures—that enable each participant to sign network packets in such

4. The mutual assistance provisions of the European Convention on Cybercrime would provide such a framework if ratified.

5. We acknowledge that some strong anonymity advocates may view the mere existence of multiple groups to be a threat to individuals' privacy, particularly for groups with few members.

6. Moreover, it may even be possible with certain group signature schemes [15] to apply threshold techniques to create the group keys themselves, thereby ensuring no party ever learns the group manager private key.

a way that: (1) an authorized party can determine the physical identity of hardware originating those packets; (2) no other party can determine the identity of the originating physical hardware; and simultaneously (3) all network participants can verify that a packet is well-formed and attributable by the trusted authority. We have shown that the technique, while still some distance from being practicable, is a viable and promising foundation for future research.

Acknowledgements

The authors are indebted to Hovav Shacham for his advice and comments on earlier drafts of this manuscript. This work is funded in part by the NSF under grants CNS-0627157 and CNS-0722031.

References

[1] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Holding the Internet Accountable. In *Proc. of the Workshop on Hot Topics in Networking (Hotnets-VI*, Atlanta, CA, 2007.

[2] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol. In *Proceedings of the ACM SIGCOMM Conference*, Seattle, CA, Aug. 2008.

[3] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, Santa Barbara, CA, USA, Aug. 20–24, 2000. Springer-Verlag, Berlin, Germany.

[4] G. Ateniese and G. Tsudik. Some open issues and directions in group signatures. In *Financial Cryptography*, 1999.

[5] G. Ateniese, G. Tsudik, and D. Song. Quasi-efficient revocation of group signatures. In *Financial Cryptography*, 2002.

[6] D. Aucsmith. The Digital Crime Scene: A Software Prospective. In *Proc. of the CyberCrime and Digital Law Enforcement Conference*, Yale Law School, Mar. 2004.

[7] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*. Springer-Verlag, Berlin, Germany, May 11–15, 1997.

[8] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629, Warsaw, Poland, May 4–8, 2003. Springer-Verlag, Berlin, Germany.

[9] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, Nov. 3–5, 1993. ACM Press.

[10] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, San Francisco, CA, USA, Feb. 14–18, 2005. Springer-Verlag, Berlin, Germany.

[11] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *ACM Computer Communications Review*, 19(2), 1989.

[12] R. Beverly and S. Bauer. The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet. In *Proc. of the Workshop on Steps to Reducing Unwanted Traffic on the Internet*, Cambridge, MA, July 2005.

[13] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55, Santa Barbara, CA, USA, Aug. 15–19, 2004. Springer-Verlag, Berlin, Germany.

[14] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *ACM CCS*, 2004.

[15] X. Boyen and B. Waters. Compact group signatures without random oracles. Cryptology ePrint Archive, Report 2005/381, 2005. http://eprint.iacr.org/.

[16] E. Brickell. An efficient protocol for anonymously providing assurance of the container of a private key, 2003. Submission to the Trusted Computing Group.

[17] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and applications to efficient revocation of anonymous credentials. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*. Springer-Verlag, Berlin, Germany, 2002.

[18] M. Carson and D. Santay. NIST Net: A Linux-Based Network Emulation Tool. 33(3):111–126, July 2003.

[19] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265, Brighton, UK, Apr. 8–11, 1991. Springer-Verlag, Berlin, Germany.

[20] D. Clark, K. Sollins, J. Wroclawski, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proc. of the ACM SIGCOMM Conference*, Aug. 2002.

[21] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *CRYPTO 2005*, LNCS. Springer-Verlag, Berlin, Germany, 2005.

[22] M. de Vivo, E. Carrasco, G. Isern, and G. O. de Vivo. A Review of Port Scanning Techniques. *ACM Computer Communications Review*, 29(2), Apr. 1999.

[23] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[24] P. Francis and R. Gummadi. IPNL: A NAT-extended Internet Architecture. In *Proc. of the ACM SIGCOMM Conference*, pages 69–80, San Diego, CA, Aug. 2001.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310, Prague, Czech Republic, May 2–6, 1999. Springer-Verlag, Berlin, Germany.

[26] A. Ghosh, L. Wong, G. D. Crescenzo, and R. Talpade. InFilter: Predictive Ingress Filtering to Detect Spoofed IP Traffic. In *Proc. of the Second International Workshop on Security in Distributed Computing Systems*, Washington, DC, 2005.

[27] R. Golding, P. Bosch, and J. Wilkes. Idleness is Not Sloth. Technical Report HPL-96-140, HP, Oct. 1996.

[28] M. Gritter and D. R. Cheriton. An Architecture for Content Routing Support in the Internet. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 37–48, San Francisco, CA, Mar. 2001.

[29] B. A. Howell. Real World Problems of Virtual Crime. In *Proc. of the CyberCrime and Digital Law Enforcement Conference*, Yale Law School, Mar. 2004.

[30] International Telecommunications Union. Traceback use cases and requirements. `http://politechbot.com/docs/itu.traceback.use.cases`.

requirements.091108.txt.
[31] C. Jin, H. Wang, and K. Shin. Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic. In *Proc. of the 10th ACM Conference on Computer and Communications Security*, Oct. 2003.
[32] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *EUROCRYPT 2004*, 2004.
[33] A. Kiayias and M. Yung. Group signatures: Provable security, efficient constructions and anonymity from trapdoor-holders. In *Mycrypt 2005, First International Conference on Cryptology in Malaysia*, 2005.
[34] J. Kirk. 'Evil twin' Wi-Fi access points proliferate. http://www.networkworld.com/news/2007/042507-infosec-evil-twin-wi-fi-access.html, Apr. 2007.
[35] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.
[36] B. Lampson. Practical Principles for Computer Security. Software System Reliability and Security, Proceedings of the 2006 Marktoberdorf Summer School, 2006.
[37] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. SAVE: Source Address Validity Enforcement Protocol. In *Proc. of the IEEE Infocom Conference*, New York, NY, 2002.
[38] H. Lipson. Tracking and Tracing Cyber-Attacks: Technical Challenges and Global Policy Issues. Technical Report CMU/SEI-2002-SR-009, CERT Coordination Center, November 2002.
[39] X. Liu, X. Yang, D. Weatherall, and T. Anderson. Efficient and Secure Source Authentication with Packet Passports. In *Proc. of the Workshop on Steps to Reducing Unwanted Traffic on the Internet*, San Jose, CA, July 2006.
[40] B. Lynn. Pairing-based cryptography library, 2006. http://crypto.stanford.edu/pbc/.
[41] D. Moore, G. M. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Proceedings of the USENIX Security Symposium*, pages 9–22, Washington, D.C., Aug. 2001.
[42] D. K. Mulligan, K. McEldowney, J. M. Garry, and B. Givens. Supplement to Intel Complaint, Potential Harm to Individuals. Center for Democracy and Technology filing with the Federal Trade Commision, Apr. 1999.
[43] NIST Computer Security Devision. Encryption Key Recovery. http://csrc.nist.gov/keyrecovery/clip.txt.
[44] C. M. E. Painter. Tracing in Internet Fraud Cases: PairGain and NEI Webworld. United States Attorney's Bulletin, May 2001.
[45] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. In *Proc. of the ACM SIGCOMM Conference*, Aug. 2006.
[46] A. M. Rutkowski. Cybercrime in New Network Ecosystem: vulnerabilities and new forensic capabilities. In *Proc. of the CyberCrime and Digital Law Enforcement Conference*, Yale Law School, Mar. 2004.
[47] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proc. of the ACM SIGCOMM Conference*, Aug. 2000.
[48] S. Schiesel. Growth of Wireless Internet Opens New Path for Thieves. New York Times, Mar. 2005.
[49] R. Security. Wireless adoption increases, security improves in world's major cities. RSA Security Press Release, May 2006.
[50] S. Shalunov. TCP over WAN Performance Tuning and Troubleshooting. http://shlang.com/writing/tcp-perf.html.
[51] W. Simpson. IP in IP Tunneling. RFC 1853, Internet Engineering Task Force, Oct. 1995.
[52] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer. Single-packet IP traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734, Dec. 2002.
[53] D. X. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proc. of the IEEE Infocomm Conference*, Apr. 2001.
[54] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, Dec. 2004.
[55] D. Wendlandt, D. G. Andersen, and A. Perrig. FastPass: Providing First-Packet Delivery. Technical Report CMU-Cylab-05-004, Carnegie Mellon University, Mar. 2006.
[56] H. Xia and J. Brustoloni. Detecting and Blocking Unauthorized Access in Wi-Fi Networks. In *Proc. of the IFIP-TC6 Networking Conference*, Athens, Greece, May 2004.
[57] A. Yaar, A. Perrig, and D. Song. FIT: Fast Internet Traceback. In *Proc. of the IEEE INFOCOM Conference*, 2005.
[58] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proc. 9th USENIX Security Symposium*, Aug. 2000.

BBS Group Signature Details

The BBS short group signature scheme [13] is specified with respect to a bilinear group pair (G_1, G_2), where G_1 and G_2 are both multiplicative cyclic groups of prime order p, and where there exists a computable isomorphism ψ from G_2 to G_1. Let $e\colon\ G_1 \times G_2 \to G_T$ denote a bilinear map. For security, we assume that the "Strong Diffie-Hellman" assumption holds on (G_1, G_2) and the "Linear Diffie-Hellman" assumption holds on G_1; we defer details to [13].

Setup. On input the number of members of the group n, KeyGen first selects a generator g_2 in G_2 uniformly at random and sets g_1 to $\psi(g_2)$. The KeyGen algorithm then sets $h \xleftarrow{\$} G_1 - \{1_{G_1}\}$ and $\xi_1, \xi_2 \xleftarrow{\$} \mathbb{Z}_p^*$, and sets $u \leftarrow h^{1/\xi_1}$ and $v \leftarrow h^{1/\xi_2}$. Then KeyGen selects $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $w \leftarrow g_2^{\gamma}$. The global public key then becomes $\mathsf{pk} \leftarrow (g_1, g_2, h, u, v, w)$.

Next, for $i = 1$ to n, KeyGen selects $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $A_i \xleftarrow{\$} g_1^{1/(\gamma + x_i)}$. The i-th user's private key becomes $\mathsf{sk}[i] \leftarrow (A_i, x_i)$, and the master secret key becomes $\mathsf{msk} \leftarrow (\xi_1, \xi_2, A_1, A_2, \ldots, A_n)$.

Sign. The Sign algorithm, on input the group public key $\mathsf{pk} = (g_1, g_2, h, u, v, w)$, a signer's secret key $\mathsf{sk}[i] = (A_i, x_i)$, and a message $M \in \{0,1\}^*$, proceeds as follows.

First, the signer picks $\alpha, \beta, r_\alpha, r_\beta, r_x, r_{\delta_1}, r_{\delta_2} \xleftarrow{\$} \mathbb{Z}_p$

and then computes

$$T_1 \leftarrow u^{\alpha} \quad (2)$$
$$T_2 \leftarrow v^{\beta} \quad (3)$$
$$T_3 \leftarrow A_i h^{\alpha+\beta} \quad (4)$$
$$R_1 \leftarrow u^{r_\alpha} \quad (5)$$
$$R_2 \leftarrow v^{r_\beta} \quad (6)$$
$$R_3 \leftarrow e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}} \quad (7)$$
$$R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\delta_1}} \quad (8)$$
$$R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\delta_2}} \ . \quad (9)$$

Let $H\colon \{0,1\}^* \rightarrow \mathbb{Z}_p$ be a hash function, which the security proofs for the BBS scheme model as a random oracle. The Sign algorithm then computes

$$c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \quad (10)$$

and then proceeds to compute

$$s_\alpha \leftarrow r_\alpha + c\alpha \quad (11)$$
$$s_\beta \leftarrow r_\beta + c\beta \quad (12)$$
$$s_x \leftarrow r_x + cx_i \quad (13)$$
$$s_{\delta_1} \leftarrow r_{\delta_1} + c\alpha x_i \quad (14)$$
$$s_{\delta_2} \leftarrow r_{\delta_2} + c\beta x_i \ . \quad (15)$$

The resulting signature is

$$\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}) \ .$$

Verify. On input the group public key $\mathsf{pk} = (g_1, g_2, h, u, v, w)$, a message M, and a signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$, the Verify algorithm computes

$$R'_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c} \quad (16)$$
$$R'_2 \leftarrow v^{s_\beta} \cdot T_2^{-c} \quad (17)$$
$$R'_3 \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot (e(T_3, w)/e(g_1, g_2))^{c} \quad (18)$$
$$R'_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}} \quad (19)$$
$$R'_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}} \quad (20)$$
$$c' \leftarrow H(M, T_1, T_2, T_3, R'_1, R'_2, R'_3, R'_4, R'_5) \quad (21)$$

If $c = c'$, Verify accepts, else it rejects.

Open. The Open algorithm takes as input the group public key $\mathsf{pk} = (g_1, g_2, h, u, v, w)$, the master secret key $\mathsf{msk} = (\xi_1, \xi_2, A_1, A_2, \ldots, A_n)$, a message, and a signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. After first using Verify to check the validity of the signature, Open computes $A' \leftarrow T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2})$ and returns the index i for which $A' = A_i$; the integer i identifies the original signer.

Revocation. Recall that $\mathsf{pk} = (g_1, g_2, h, u, v, w)$ is the group public key. To revoke the r-th user, with private key (A_r, x_r), an authority publishes the revocation list $\mathsf{RL} = \{(A_r^*, x_r)\}$, where $A_r^* = g_2^{1/(\gamma + x_r)}$. Note that $A_r = \psi(A_r^*)$. When $G_1 = G_2$, $A_r^* = A_r$; otherwise, the computation of A_r^* uses the secret γ from the setup procedure.

Given RL, verifiers first compute the new public key $\mathsf{pk}' = (g'_1, g'_2, h, u, v, w')$ as: $g'_1 \leftarrow \psi(A_r^*)$, $g'_2 \leftarrow A_r^*$, and $w' \leftarrow g_2 \cdot (A_r^*)^{-x_r}$. An unrevoked user with private key (A, x) sets its new private key to be (A', x) where

$$A' = \psi(A_r^*)^{1/(x - x_r)} / A^{1/(x - x_r)} \ .$$

If $m > 1$ users are being revoked, the above process could be executed once for each of the m revoked users, or m users could be revoked in batch; we omit details.